

Root ORAM: A Tunable Differentially Private Oblivious RAM

Sameer Wagh
Princeton University
swagh@princeton.edu

Paul Cuff
Princeton University
cuff@princeton.edu

Prateek Mittal
Princeton University
pmittal@princeton.edu

ABSTRACT

State-of-the-art mechanisms for oblivious RAM (ORAM) suffer from significant bandwidth overheads (greater than 100x) that impact the throughput and latency of memory accesses. This renders their deployment in high-performance and bandwidth-constrained applications difficult, motivating the design of low-overhead approaches for memory access obfuscation.

In this work, we introduce and formalize the notion of a differentially private ORAM that provides statistical privacy guarantees, and which to the extent of our knowledge, is the first of its kind. The formalization of differentially private ORAM opens up a large design space of low-bandwidth ORAM protocols that can be deployed in bandwidth constrained applications.

We present Root ORAM, a family of practical ORAMs that provide a tunable, multi-dimensional trade-off between the desired bandwidth overhead, *outsourcing ratio*¹ and the system security, and that provide rigorous privacy guarantees of differentially private ORAMs. The Root ORAM protocols can be tuned to achieve application-specific bandwidth constraints, enabling practical deployment, at the cost of statistical privacy guarantees quantified under the differential privacy framework and lower outsourcing ratios.

We demonstrate the practicality of Root ORAM using theoretical analysis, simulations, as well as experiments on Amazon EC2. Our theoretical analysis rigorously quantifies the privacy offered by Root ORAM, and provably bounds the information leaked from observing memory access patterns. Our experimental analysis shows the feasibility of these protocols using realistic simulations. The simplest protocol in the Root ORAM family requires a bandwidth of mere 10 blocks, at the cost of rigorously quantified security loss and a low outsourcing ratio. This is an order of magnitude improvement over the existing state-of-the-art ORAM schemes which incur logarithmic bandwidth overheads.

¹*Outsourcing ratio* is defined as the amount of data that can be outsourced for a certain amount of local storage. Refer to Sec. 7 for more details.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '16 October 24–28, 2016, Hofburg Palace, Vienna, Austria

© 2016 ACM. ISBN 978-1-4503-2138-9...\$15.00

DOI: 10.1145/1235

1. INTRODUCTION

Cloud storage and computing are important tools to outsource data but have given rise to significant privacy concerns due to the non-local nature of data storage. Though encryption goes a long way in assuring data confidentiality, recent work [5, 12] has shown that encryption is not sufficient. Encryption does not hide memory access patterns; an untrusted storage server can thus perform traffic analysis of memory access patterns to compromise client privacy. The work of Islam *et al.* has shown the leakage of sensitive keyword information by performing traffic analysis of access patterns over encrypted email [12]. Similarly, Dautrich *et al.* have shown that access patterns over database tuples can leak ordering information [5].

Oblivious RAM (ORAM), first introduced by Goldreich and Ostrovsky [10, 11], is a cryptographic primitive which allows a client to protect its data access pattern from an untrusted server storing the data. Since its introduction, substantial progress has been made by the research community in developing novel and efficient ORAM schemes [4, 9, 16, 17, 20–22]. Recent work has also shown the promise of using ORAMs as a critical component in developing protocols for cryptographic primitives such as Secure Multi-Party Computation [9].

However, ORAM schemes incur a large overhead in terms of bandwidth that renders them impractical. For example, even the most efficient ORAM protocols [17, 21, 22] incur a logarithmic overhead compared to conventional RAMs (greater than 100x including constants). This significantly impacts the throughput and latency of memory accesses, and presents a bottleneck for real-world deployment of ORAMs in high-performance and bandwidth constrained applications. The lack of low-bandwidth ORAMs, despite considerable efforts from the security community, is an undeniable indicator for the need of a fundamentally new approach.

Hence, we propose a novel approach for developing practical ORAM protocols that can support even a constant bandwidth overhead compared to conventional RAMs. Our key approach is to provide statistical privacy guarantees and propose tunable protocol designs. We first formalize the notion of a *differentially private ORAM* that provides statistical privacy guarantees, and which to the extent of our knowledge, is the first of its kind. As the name suggests, we use the differential privacy framework developed by Dwork *et al.* [6] with its (ϵ, δ) -differential privacy modification [7]. In the current formulation of an ORAM, the output is computationally indistinguishable for any two input sequences. In a differentially private ORAM, we characterize the effect

of a small change in the ORAM input to the change in the probability distribution at the output.

We also present Root ORAM², a family of ORAM schemes that provide tunable ORAM protocols allowing variable bandwidth overheads, system security and outsourcing ratios and including a design point that supports constant bandwidth construction and provide rigorous privacy guarantees of differentially private ORAMs. The low bandwidth protocols, achieved at the cost of statistical privacy and lower outsourcing ratios, are an order of magnitude improvement over previous work in which the protocols still incur a logarithmic bandwidth [9, 21, 22].

The formalization of a differentially private ORAM opens up a large underlying design space currently not considered by the community. With rigorously quantified privacy guarantees, we propose Root ORAM as the first step in the direction of statistically private ORAMs.

1.1 Our Contributions

Root ORAM introduces a number of paradigm shifts in the design of ORAM protocols while at the same time building on the prevailing ideas of contemporary ORAM constructions³. Our main contributions can be summarized as follows:

The notion of a *differentially private ORAM*: We formalize the notion of a *differentially private ORAM*, which to the extent of our knowledge is the first of its kind. Formally discussed in Section 3, a differentially private ORAM bounds the information leakage from memory access patterns of an ORAM protocol.

Tunable/parametric protocol family: In bandwidth constraint applications, large bandwidth overhead ($>100\times$) of conventional ORAM schemes is a significant bottleneck. We propose the tunable reduction of bandwidth with outsourcing ratio and a rigorously quantified privacy loss. We propose a new family of ORAM schemes called Root ORAM which can be tailored as per the needs and constraints of the application to achieve desired security and bandwidth. This serves as a key enabler for practical deployment and is discussed in more detail in Section 4.5.

Security: We analyze and provide theoretical guarantees for the security offered by Root ORAM schemes in the new differentially private ORAM framework. As a consequence of our analysis, we also give a new proof of the Path ORAM protocol security in the new framework. Our approach is general and will be useful for rigorously reasoning about the security of alternative statistically private ORAM schemes in the future. This is presented in Section 6.

Performance: Root ORAM introduces a new design point with constant bandwidth overhead. The simplest protocol of the family has bandwidth usage per access as low as a constant around 10 data blocks⁴ compared to $10 \cdot \log N$ blocks in the case of Path ORAM. But this comes at the cost

²The protocol family is called Root ORAM because in the lowest bandwidth regime, the data structure reduces to just the root and the leaves (tree of depth 1). Refer to Sec. 5 for details.

³This work is inspired by the Path ORAM paper [22] and we would like to give the authors of the paper all due credit. At the same time, we would like to highlight the fact that there are considerable differences between the two protocols (as given in section 4).

⁴Achieved for $\lambda = 4$ and $Z = 2$. Refer to Section 5 for details.

a rigorously quantified security loss as well as a reduction in outsourcing ratio. At the same time, the server-side storage efficiency can be as high as 1:2 i.e., one dummy block per real block outsourced (compared to Path ORAM which uses around 1:10). We implement Root ORAM and demonstrate its practicality using simulations as well as real world experiments on Amazon EC2. These results are covered in Section 7.

These contributions are order of magnitude improvement over state-of-the-art protocols, though we would like to remind the reader that these come at the cost of a rigorously quantified privacy loss and a reduction in outsourcing ratio. Finally, *Root ORAM does not assume any server-side computation* and uses low, practical amounts of client-side storage at the same time being extremely simple to implement at both the client and the server side.

2. PRELIMINARIES: STATISTICAL PRIVACY

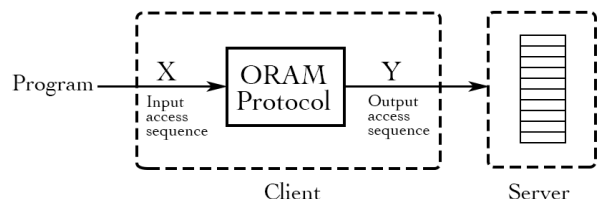


Figure 1: **A representation of an ORAM. The protocol box translates an input access sequence into an output access sequence.**

The significant bandwidth overhead in conventional ORAM schemes despite considerable research efforts necessitates a paradigm shift in our approach to protocol design. To this extent, we formulate the concept of statistical privacy in ORAMs.

Here we present a brief overview of the notion of statistical privacy in the ORAM context. A perfect ORAM roughly⁵ leaks no information about the input access sequence. In other words, we can consider an ORAM to be a black-box with an input sequence as X and an output sequence as Y as shown in Fig. 1. An ORAM with perfect privacy would guarantee the independence of X and Y . A slightly stronger condition could be to say that the distribution of the output sequence is uniform over its space for any given input⁶.

The most natural way to extend the latter condition for designing statistically private ORAMs is to consider ORAM schemes that give non-uniform distributions of the output sequences Y (for a given input X) and use security metrics that quantify the “non-uniformity” of this distribution. This is graphically illustrated in Fig. 2, where an attacker aims to guess the original access pattern after observing the output access pattern o .

⁵We say roughly because ORAMs could leak information such as timing of accesses/access pattern size.

⁶This is stronger because conventional definitions of perfect ORAMs involve outputs being computationally indistinguishable which hides a small detail which we shall see in Sec. 6.

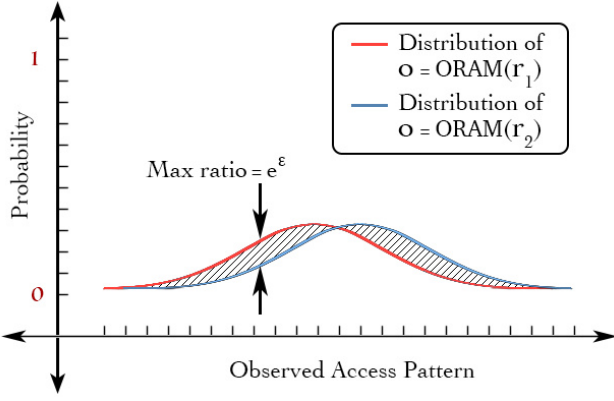


Figure 2: This figure shows the intuition behind statistically private ORAMs. In particular, differential privacy based statistical ORAM.

3. DIFFERENTIALLY PRIVATE ORAM

The notion of statistical privacy is certainly not new across security/privacy applications in current literature [1, 13]; but it has never been previously explored in the context of ORAMs. We believe formulating such a framework would greatly expand the ability of the research community to develop novel ORAM protocols with low-bandwidth overhead, serving as an enabler for real-world deployment of this technology.

Over the years, a number of papers have been published in the ORAM domain which adopt quite a few different definitions to quantify ORAM bandwidth overhead. We will use the original and straightforward definition of bandwidth as the average number of blocks transferred for one access [4].

DEFINITION 1. *The bandwidth cost of a storage scheme is given by the average number of blocks transferred in order to read or write a single block.*

Formally, an ORAM is defined as a mechanism (possibly randomized) which takes an input access sequence \vec{y} as given below,

$$\vec{y} = ((\text{op}_M, \text{addr}_M, \text{data}_M), \dots, (\text{op}_1, \text{addr}_1, \text{data}_1)) \quad (1)$$

and outputs a resulting output sequence denoted by $\text{ORAM}(\vec{y})$. Here, M is the length of the access sequence, op_i denotes whether the i^{th} operation is a read or a write, addr_i denotes the address for that access, and data_i denotes the data (if op_i is a write). Denoting by $|\vec{y}|$ the length of the access sequence \vec{y} , the currently accepted security definition for ORAM security can be summarized as follows [22]:

DEFINITION 2. (Currently accepted ORAM Security) : *Let \vec{y} as given in Eq. 1, denote an input access sequence. Let $\text{ORAM}(\vec{y})$ be the resulting randomized data request sequence of an ORAM algorithm. The ORAM protocol guarantees that for any \vec{y} and \vec{y}' , $\text{ORAM}(\vec{y})$ and $\text{ORAM}(\vec{y}')$ are computationally indistinguishable if $|\vec{y}| = |\vec{y}'|$, and also that for any \vec{y} the data returned to the client by ORAM is consistent with \vec{y} (i.e the ORAM behaves like a valid RAM) with high probability.*

This framework for ORAMs is constructed with complete security at its core [4, 9, 17, 22] and there is no natural way to extend this to incorporate a statistical privacy notion.

Hence, we introduce and formalize the following statistical notion of an ORAM: *differentially private ORAM*.⁷

3.1 Formalizing DP-ORAM

The intuition behind a differentially private ORAM is that given any two input sequences that differ in a single access, the distributions of their output sequences should be “close”. In other words, similar access sequences lead to similar distributions. We formally define it as follows:

DEFINITION 3. Differentially Private ORAM : *Let \vec{y} , as defined in Eq. 1, denote the input to an ORAM. Let $\text{ORAM}(\vec{y})$ be the resulting randomized data request sequence of an ORAM algorithm. We say that a ORAM protocol is (ϵ, δ) -differentially private if for all input access sequences \vec{y}_1 and \vec{y}_2 , which differ in at most one access, the following condition is satisfied by the ORAM protocol,*

$$\Pr[\text{ORAM}(\vec{y}_1) \in S] \leq e^\epsilon \Pr[\text{ORAM}(\vec{y}_2) \in S] + \delta \quad (2)$$

where S is any set of output sequences of the ORAM.

We note that the formalism does not make any assumption about the size of the output sequences in S . Thus, if the input to the ORAM is changed by a single access tuple $(\text{op}_i, \text{addr}_i, \text{data}_i)$, the output distribution does not change significantly. Fig. 2 graphically represents this intuition. Given two sequences r_1 and r_2 , the two distributions generated (the red and the blue) are close to each other in the differential privacy sense.

It is important to note that the differential privacy guarantees when two access patterns differ in multiple elements directly follows from the composability property of differential privacy. Since this property is extremely important for the utility of the mechanism, we summarize this in the form of a theorem.

Theorem 1 (Composability of DP-ORAM). *Given two access sequences s_1 and s_2 that differ in m accesses, a (ϵ, δ) -differentially private ORAM mechanism guarantees,*

$$\Pr[\text{ORAM}(s_1) \in S] \leq e^{m\epsilon} \Pr[\text{ORAM}(s_2) \in S] + m\delta \quad (3)$$

The proof of the theorem directly follows from the composability property of the differential privacy mechanism [15]. In other words, Root ORAM guarantees can be extended to sequences which differ in multiple accesses and hence can be used to give rigorous guarantees for arbitrary access sequences.

4. ROOT ORAM OVERVIEW

In this section, we briefly describe our key design goals and give a high level overview of the Root ORAM protocol. The notation used is given in Table 1.

4.1 Design Goals

Tunable ORAM scheme: We target a tunable architecture with explicit low bandwidth regimes which can be used to design ORAM protocols for bandwidth constrained applications.

⁷For the rest of the paper, we shall use DP-ORAM to mean a differentially private ORAM

Symbol	Description
$N = 2^L$	Number of real data blocks outsourced
$k \geq 1$	Model parameter (to tune bandwidth)
$p \in (0, 1 - 1/N]$	Model parameter (to tune security)
Z	Number of blocks in each bucket
B	Size of each block (in bits)
$P(x)$	Path from leaf x to the root
$P(x, i)$	Node at level i in $P(x)$
$x := \text{position}[a]$	Data block a is currently mapped to leaf x i.e. a resides in some bucket in $P(x)$

Table 1: Notation for Root ORAM

Secure framework: We target protocols that provide rigorous privacy guarantees viz. that of differentially private ORAMs formalized in Section 3.

Low Storage and Computation: The design should use as low storage as possible both on the client as well as the server side. *Server side computation is not always practical and hence we would like to avoid assuming any such capability.*

4.2 Approach Overview

Root ORAM protocol can be split into three components, the access, the new mapping and the eviction. These are briefly described below. As Path ORAM is an instantiation of Root ORAM, the protocols are very similar in their structure but have important subtle differences.

Since we demonstrate the feasibility of having a tunable ORAM architecture, we just need to show the existence of such a trade-off. *In particular, we do not claim this construction to be the optimal way of achieving trade-offs.* We choose the server-side storage to be stored in a partial binary tree where each node is a bucket which can hold up to Z data blocks. A stash at the client is used to store a small amount of data. Data elements are mapped to leaves and this mapping is stored locally.

Access: The main invariant (same as Path ORAM) is that any data block is along the path from the root to the leaf it is mapped or is in the stash. To access a data element, the client looks up the local mapping to find the leaf that the data element is mapped onto.

New Mapping: The relevant data block is then read or written with the new data and a new mapping is generated. It is important to note that this new mapping is not uniform among the leaves. There is tremendous flexibility in choosing this distribution; for our purposes, we choose the distribution to be given by Eq. 4, i.e., the new mapping is slightly more likely to be the same as the old mapping than any other random leaf.

Finally, new randomized encryptions are generated and all the data is written back with elements being pushed down further in the tree if possible (towards the leaf) and if new elements can be written back to the tree.

Stash Recursion and Eviction: *Root ORAM uses the recursion technique on the local stash.* It also uses fake accesses to keep a low stash value. The client machine independently sends fake access queries to the server, completely indistinguishable from normal requests, through a Poisson process with parameter λ . The eviction process and the recursion together ensure low stash size.

4.3 Comparison with Path ORAM [22]

Root ORAM is inspired by the Path ORAM protocol and

we would like to give the authors all due credit. At the same time, in this subsection, we would like to highlight the critical differences between the two papers.

Differentially Private ORAM: Root ORAM introduces a new rigorous metric to quantify ORAM security, which extends current formalism to include the notion of a statistically private ORAM. We rigorously bound the privacy offered by the Root ORAM (as well as Path ORAM) using this metric.

Storage structure: Root ORAM uses a partial binary tree as the storage structure at the server where the height of the tree is a model parameter k . This is represented in Fig 3. The parameter k governs the bandwidth of the protocol. The Path ORAM protocol on the contrary has a fixed height binary tree (complete binary tree).

Tunability: The ability to tune the protocol as per the system constraints is a stark difference between Root ORAM and Path ORAM. *There is no way to optimize Path ORAM when the bandwidth is constrained and statistical security is acceptable.* Root ORAM introduces the novel notion of non-uniform mapping and gives rigorous statistical privacy guarantees. Path ORAM’s update mapping scheme then turns out to be a special case of this mapping.

Simply by tuning the parameters, Root ORAM matches or exceeds the performance of Path ORAM. We provide the ability to operate in the low bandwidth regime which Path ORAM cannot support. The eviction scheme allows Root ORAM to achieve perfect security ($\epsilon = 0$) at even lower bandwidth than the Path ORAM protocol.⁸

Eviction scheme: The eviction schemes of the two protocols have subtle differences. Path ORAM relies on a sufficiently large bucket size to achieve its goals. In contrast, Root ORAM uses an eviction scheme of fake accesses. *Root ORAM parameters can be tuned to achieve Path ORAM protocol, but the latter is not the lowest bandwidth full security ($\epsilon = 0$) protocol in the Root ORAM family.*

Multi-dimensional design space: Root ORAM can be tuned as per the user’s requirements in terms of the security desired, the bandwidth available and local storage required. Thus, Root ORAM offers attractive design points that can support a wide range of operating conditions just by tuning its parameters.

5. ROOT ORAM DETAILS

In this section, we provide the details of Root ORAM. We begin by describing the basics of the protocol. The required notation is tabulated in Table 1.

5.1 Server Storage

Server Storage: The server stores data in the form of a partial binary tree consisting of buckets⁹ as nodes. In other words, given an integer k , we first construct a binary tree of depth k i.e., root at level 0 and the lowest level at level $k - 1$ (this will have 2^{k-1} leaves). Then each of the 2^{k-1} leaves of this tree has 2^{L-k+1} children each (From here on, we shall refer to these $N = 2^L$ nodes as the *leaves* of the tree). This set-up is illustrated in Fig. 3.

⁸For $\epsilon = 0$, Path ORAM uses a bandwidth of $\sim 10 \log N$ data blocks per access whereas Root ORAM can perform the same with around $\sim 8 \log N$ for the following choice of parameters: $Z = 2, \lambda = 1, p = 1 - 2^{-k}$.

⁹A bucket contains multiple blocks of data storage which can be either real or dummy.

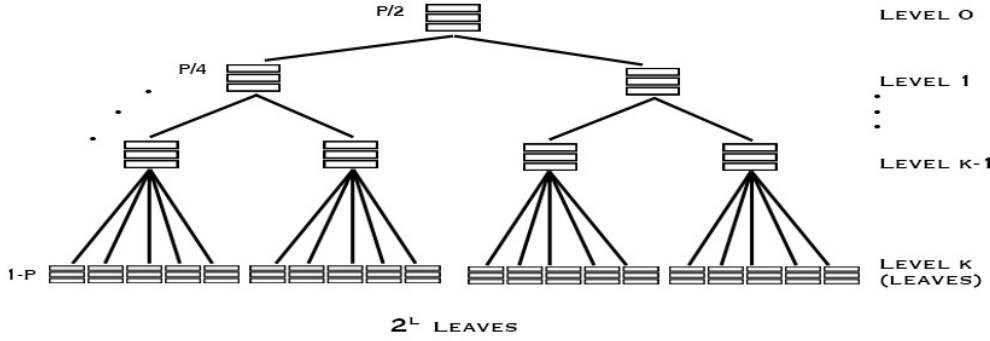


Figure 3: *Root ORAM server storage* : The figure illustrates the server side storage. The level 0 to $k - 1$ form a binary tree and the last level of the tree contains $N = 2^L$ leaves evenly distributed over the binary tree leaves.

Bucket structure: Each node is a bucket consisting of Z blocks, each block can either be real or dummy (encryptions of 0).

Path structure: The leaves are numbered in the set $\{0, 1, \dots, 2^L - 1\}$. $P(x)$ denotes the path (set of buckets along the way) from leaf x to the root and $P(x, i)$ denotes the bucket in $P(x)$ at level i . It is important to emphasize here that the path length in Root ORAM is $(k + 1)$ blocks compared to the $(\log N) + 1$ in Path ORAM.

Dummy blocks and randomized encryption: We use the standard padding technique (fill buckets with dummy blocks when needed) along with randomized encryption to ensure indistinguishability of real and dummy blocks.

5.2 Invariants of the scheme

Main Invariant (same as Path ORAM) : The main invariant in Root ORAM is that each *real* data block a is mapped to a leaf $x := \text{position}[a]$, $x \in \{0, 1, 2, \dots, 2^L - 1\}$ and at any point in the execution of the ORAM, the real block will be somewhere in a bucket $\in P(x)$ or in the local Stash.¹⁰

Secondary Invariant : We maintain the secondary invariant that after each access to an element, its new mapping is governed by a constant but *non-uniform distribution* D . There is tremendous flexibility in choosing this distribution; for our purposes, we consider the distribution D given by the following equation and shown graphically in Fig. 4.

$$P_{z,x} = p_2 + (p_1 - p_2)\delta_{zx} \quad (4)$$

Where $P_{z,x}$ is the probability that an element accessed from leaf x is mapped to a leaf z , δ_{ij} is the Kronecker¹¹ delta, $p_1 = (1 - p)$ and $p_2 = p/(N - 1)$ where p is the model parameter as defined in Table. 1.

5.3 Client Storage

Position Map: The client side stores a position map which maps real data blocks to leaves of the server tree.

¹⁰It is important to note that the invariant does not say that the position each data block is uniform over the set of leaves, as shall be clarified by the second invariant.

¹¹Kronecker delta is defined as

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

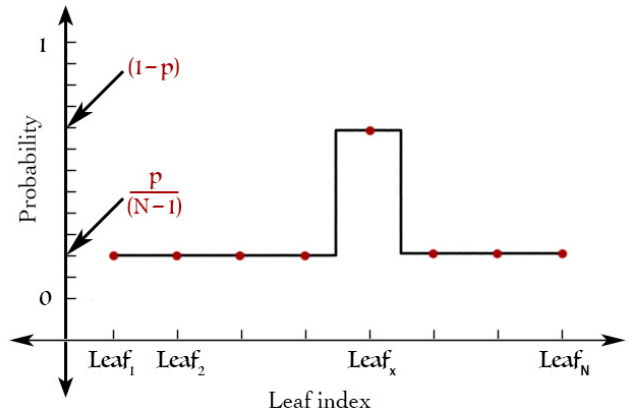


Figure 4: The new position of a data block is in general non-uniform according to this distribution. The distribution reduces to uniform when $p = 1 - 2^{-L}$

Stash: The client maintains a local stash, which is a small amount of storage locally at the client which is used to store overflow data blocks locally.

Recursion: The client position map and the stash are both stored recursively. The recursion technique has been used in previous works [19, 21, 22]. The idea is to use another ORAM to store the position map at the server side instead of storing directly at the client size and recurse. The position map for the final ORAM is stored locally.

5.4 Main idea

The main idea of the protocol is very simple, we read data along a path, try to write data back to the same path (with some modifications and new encryptions) and if there is insufficient storage, we retain those overflow data elements back in the local Stash.

Along with this, there is an independent access process of fake accesses¹². These accesses are made by the user to the server and are indistinguishable from real accesses. Fake accesses are drawn from a Poisson process with a parameter λ . It is important to note that in Root ORAM, the real

¹²This is similar to the eviction scheme described in [18, 19] with the crucial difference that our fake accesses are completely indistinguishable from real accesses.

access by the user and the fake accesses by the client machine are exactly the same and hence are indistinguishable from the server's perspective.

5.5 Details of the protocol

An access is defined as a 3-tuple

$$\text{Access}_i = (\text{data}_i, \text{element}_i, \text{operation}_i)$$

For a real access, given a particular access 3-tuple, the user finds the mapping of the data block needed using his local position map. He then requests the whole path of that leaf from the server tree. After processing the data and generating new randomized encryptions, the user writes the data back to the tree with the element that was accessed at a new location along the path. But the key idea here is that the element that was accessed has a non-uniform distribution¹³ of it being mapped to other leaves. It is more likely to be mapped to the same leaf than to others and the probabilities involved are decided by the security parameter p .

The broader picture of the protocol is as follows. The client systems makes real as well as fake accesses to the server. The real access is as described in the previous paragraph. There is a parameter λ which controls the amount of fake accesses. One way of implementing the protocol is in the following way.¹⁴

normal_access(a): A normal access consists of the following functions in order: `read(a)`, `push_down(position[a])`, `update_mapping(a)` and finally a `write(a)`.

Access(op, a, data*) :

```

1: while ORAM is under use do
2:    $\alpha \leftarrow \text{Poisson}(\lambda)$ 
3:   for  $i = 1 : \alpha$  do
4:     normal_access(a)
5:   end for
6:   fake_access()
7: end while

```

read(a): The reading phase is the same as that in Path ORAM; Using the local client side mapping, the client finds out the leaf to which the data element a is currently mapped i.e find x such that $x := \text{position}[a]$. We then request all the data blocks in the buckets along path $P(x)$. The invariant ensures that the client can retrieve a , its data element, from these. This completes the reading phase of the protocol.

read(a) :

```

1:  $x \leftarrow \text{position}[a]$ 
2: for  $i \in \{0, 1, \dots, k\}$  do
3:    $S \leftarrow S \cup \text{ReadBucket}(P(x, i))$ 
4: end for

```

update_mapping(a): After reading a data block, we modify its mapping using the distribution mentioned in Eq.4 i.e `update_mapping` keeps the mapping same with probability $(1 - p)$ and with the remaining probability changes it to a uniformly random leaf among the remaining leaves.

¹³The distribution becomes uniform if $p = 1 - 2^{-L} = 1 - 1/N$.

¹⁴It should be noted that the code has been structured in the following way for clarity of understanding and hence can be optimized in a number of ways.

update_mapping(a) :

```

1:  $x \leftarrow \text{position}[a]$ 
2: if  $\text{Bernoulli}(p) = 0$  then
3:   return  $x$ 
4: else
5:   return  $\text{UniformRandom}(\{0, 1, 2, \dots, 2^L - 1\} \setminus \{x\})$ 
6: end if

```

push_down(position[a]): When any path is accessed, this function tries to place any data blocks along the path $P(\text{position}[a])$ or in the Stash to lower positions on the same path if possible.

write(a) : Once the mapping is updated, say initially $x := \text{position}[a]$ and after updating the mapping $z := \text{position}[a]$, we try to write the data block back into the bucket which is the lowest intersection of the two paths in consideration i.e. lowest bucket in $P(x) \cap P(z)$ (with the convention that bucket with the highest level number is the root at level 0) which has an empty/dummy block.

fake_access(): A fake access is issued to push back elements from the stash to the tree. One data block, say a' , is chosen at random from the stash¹⁵ and a normal access is performed on a' , i.e., `read(a')` followed by `push_down(position[a'])` followed by an `update_mapping(a')` followed by a `write(a')`.

6. THEORETICAL EVALUATION

In this section, we shall state our main theorems, their proofs and a few interesting special cases.

Theorem 2 (Main theorem). *Given a stash size C , the Root ORAM protocol with parameters k, p, Z and λ is (ϵ, δ) -differentially private for $\epsilon = 2 \log \left(\frac{(N-1)(1-p)}{p} \right)$ and $\delta = (1 - p)^{M_k}$ where $M_k = (C + Z(k + 1) + 1)$*

Proof: First, we remark that due to the conservative security analysis (Refer Sec. 6.1), λ does not appear in the above expressions. The theorem has two parts, the ϵ bound and the δ bound. Firstly, we give a brief insight into the two security parameters ϵ and δ . The proof is then structured as follows:

The ϵ bound:

- Set up the differential privacy framework for ORAM.
- Then we set up the probability evaluation model to find the probability of that a particular real sequence leads to a particular output sequence by the ORAM.
- Then we compute the maximum change that one access in the input sequence can have on the probability of the output sequence (over all output sequences).
- Finally we complete the ϵ bound.

The δ bound:

- We first show the need for δ in the security.
- We then conservatively evaluate a bound on δ .

ϵ, δ interpretation: Given an ORAM scheme with an unbounded amount of local stash, we show that such a scheme is ϵ -differentially private. But the moment we introduce a

¹⁵The security analysis takes care of the correlation of the fake access with stash elements by a worst case analysis.

Symbol	Description
M	Access pattern size
C	Stash size
p_1	$(1 - p)$
p_2	$p/(N - 1)$
M_k	$M_k = Z(k + 1) + C$

Table 2: Additional notation for security analysis

finite amount of stash, this is no longer true as is shown in Sec. 6.4. And the privacy loss under such a situation is precisely the quantity that is bounded by δ .

In the context of Path ORAM, δ characterizes the privacy loss if the stash size exceeds its bounds. Similarly, in Root ORAM, δ quantifies the privacy loss if the stash size is exceeded.

The ϵ bound

6.1 Framework set-up

The additional notation used is specified in Table 2. We conservatively assume that the adversary knows the real sequences (in other words we assume $\lambda = \infty$). Essentially, among the M access of which some are real and some are fake, we conservatively assume that these can be distinguished. In practice, the security offered by our approach is higher since the untrusted server storage cannot differentiate fake accesses from real accesses in practice.

More formally, let f_i denote the set of fake accesses and r_i denotes the real set of accesses made by the ORAM. We denote by R_i , the complete set of accesses made (r_i along with f_i). Thus we have that:

$$\max_{|r_1 - r_2| = 1} \frac{\Pr[\text{ORAM}(r_1) = o]}{\Pr[\text{ORAM}(r_2) = o]} \leq \max_{|R_1 - R_2| = 1} \frac{\Pr[\overline{\text{ORAM}}(R_1) = o]}{\Pr[\overline{\text{ORAM}}(R_2) = o]} \quad (5)$$

where $\overline{\text{ORAM}}(R_i)$ denotes the ORAM protocol output on sequence R_i without any additional fake accesses. But to prove the bounds of differential privacy in the theorem, we need to bound the following term:

$$\max_{|r_1 - r_2| = 1} \frac{\Pr[\text{ORAM}(r_1) = o]}{\Pr[\text{ORAM}(r_2) = o]} \leq e^\epsilon$$

Hence, it suffices to bound the latter term in Eq. 5 by e^ϵ .

6.2 Probability model

Next, we evaluate the ratio of the probabilities by invoking the secondary invariant. Recall that our secondary invariant is: after each access(real/fake) for an element, the position map of that element (and none other) changes randomly according to the distribution D given in Eq. 4.

With this invariant, we can compute the probability of a particular real sequence R leading to a particular observed sequence o . For our computation, we write the real sequence (including fake accesses) below the observed sequence and calculate the probabilities according to the following rules:

- The first time a data block is accessed, its location is random. Hence, we write a $1/N$ below this access.
- When an element that was accessed before is accessed, we write a p_1 or p_2 in the probability calculation depending

Observed seq.	a	b	a	c	a	a	b	d
Real seq.	x	y	x	z	y	y	z	x
Probabilities	$\frac{1}{N}$	$\frac{1}{N}$	p_1	$\frac{1}{N}$	p_2	p_1	p_2	p_2

Table 3: An example of writing probabilities given the real and observed access patterns r and o . Different symbols are used for real and observed access patterns merely for the clarity of the demonstration. p_1 and p_2 are as defined in Eq. 4 and Table 2.

Observed seq.	a	b	a	c	a	a	b	d
Real seq.	x	y	x	z	y	y	z	x
Probabilities	$\frac{1}{N}$	$\frac{1}{N}$	p_1	$\frac{1}{N}$	p_2	p_1	p_2	p_2

Table 4: Only the blue symbols affect the probability that will be written under the data element shown by an enclosing box. The red elements show the previous and next access of the boxed data element.

on whether the observed locations were same or different respectively.

- A background check is maintained, if at any time there are more than $(k + 1) \times Z + C$ data blocks mapped to the same location, the probability becomes 0, where the symbols can be found from the notations in Table 1,3. Refer to Subsection 6.4 for details.
- Finally, we multiply all the written probabilities to get the probability $\Pr[\text{ORAM}(R) = o]$.

This is demonstrated in the Table 3.

6.3 Maximum change

Next, we find the maximum change in the probabilities that can occur as a result of changing one access.

First we note that in the probability model, the probability written under each data element depends only its current observed location and its previous observed location and nothing else (and is governed by the distribution D given by Eq. 4). Hence, if one data access is changed, the maximum change that can occur in the probability is at most in two places viz, the location which was modified and the next accessed location of that data element. With this, we can enumerate all the possible cases that can occur and find the maximum change in probabilities. To do this efficiently, we develop some more notation.

Let the accessed data element be changed from **a** to **b**. Let the previous location of access of data element **a** be l_{pa} (leaf pa) and the next location be l_{na} . Similarly, the previous location of access of **b** is l_{pb} and the next location as l_{nb} . If any of these 4 do not exist i.e the symbol was never accessed before or was never accessed afterwards, we define that leaf to be 0 for simplification of the equations¹⁶. Let l be the location of the access in consideration i.e the location of data access which was changed in r_1 and r_2 . Note that in the DP-ORAM calculations, we have the same observed sequence for both the sequences r_1 and r_2 , the location of access l is the same in both the sequences. This is shown in the Fig. 5.

¹⁶In other words, if data element **a** was never accessed after the location of access change, then $l_{na} = 0$.

$o = \text{ORAM}(r_1)$	$\cdot l_{pb} \dots l_{pa} \dots l \dots l_{na} \dots l_{nb} \cdot$
r_1	$\cdot b \dots a \dots \boxed{a} \dots a \dots b \cdot$
r_2	$\cdot b \dots a \dots \boxed{b} \dots a \dots b \cdot$
$o = \text{ORAM}(r_2)$	$\cdot l_{pb} \dots l_{pa} \dots l \dots l_{na} \dots l_{nb} \cdot$

Figure 5: **The sequences r_1 and r_2 differ by one element (boxed). The previous accessed location and the next accessed location are as shown, dots are irrelevant accesses(not a or b). The observed sequence o is the same for both (DP-ORAM condition).**

Now, the probabilities can differ in at most 3 places v.i.z l , l_{na} and l_{nb} . Let r_1 be the sequence with symbol a and r_2 be the sequence with symbol b . To make the equations crisp, we define the following extension to the Kronecker delta function,

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \\ \frac{1/N - p_2}{p_1 - p_2} & \text{if } j = 0 \end{cases}$$

This modification of the Kronecker delta is for the simplicity of the equations. Specifically, the modification ensures that if a symbol is accessed for the first time, then its probability given by $P_{z,x} = p_2 + (p_1 - p_2)\delta_{zx}$ evaluates to $1/N$, as it should. Now if $\Pr[\text{ORAM}(R_1) = o] > 0$ and $\Pr[\text{ORAM}(R_2) = o] > 0$ i.e., both the ratios are well-defined, we can calculate the ratio of the probabilities as:

$$\frac{\Pr[\text{ORAM}(R_1) = o]}{\Pr[\text{ORAM}(R_2) = o]} = \frac{P_{l,lp_a} \cdot P_{l_{na},l} \cdot P_{l_{nb},l_{pb}}}{P_{l_{na},lp_a} \cdot P_{l,lp_b} \cdot P_{l_{nb},l}}$$

After observing that $\frac{1/N}{p_1} \geq \frac{p_2}{p_1}$, we can see that this maximum value of the ratio of probabilities occurs when $l_{na} = l = l_{pa}$ and $l_{pb} = l_{nb} \neq l$. In this case, the ratio is given by,

$$\frac{p_1 \cdot p_1 \cdot p_1}{p_1 \cdot p_2 \cdot p_2} = \left(\frac{p_1}{p_2}\right)^2$$

Evaluating this in terms of our parameters, $p_1 = (1 - p)$ and $p_2 = \frac{p}{N-1}$ and plugging this into the differential privacy equation, we get

$$\begin{aligned} \max_{\substack{r_1, r_2 \\ |r_1 - r_2| = 1}} \frac{\Pr[\text{ORAM}(r_1) = o]}{\Pr[\text{ORAM}(r_2) = o]} &\leq \max_{\substack{R_1, R_2 \\ |R_1 - R_2| = 1}} \frac{\Pr[\text{ORAM}(R_1) = o]}{\Pr[\text{ORAM}(R_1) = o]} \\ &\leq \left(\frac{p_1}{p_2}\right)^2 \\ &= \left(\frac{(N-1)(1-p)}{p}\right)^2 \end{aligned}$$

It is important to note that the above equation holds for all observed access sequences o . And hence, we can see that Root ORAM guarantees $\epsilon = 2 \log \left(\frac{(N-1)(1-p)}{p}\right)$. This completes the ϵ bound.¹⁷

The δ bound

¹⁷Another important point to note here is that the above analysis is a worst case analysis and hence it only depends on two probabilities in the distribution D viz., the largest and

6.4 The need for δ

In this subsection, we show the need for δ in quantifying the security. We use the Path ORAM paper to demonstrate this short-coming. We assume that the Stash size is bounded by C and let M_L denote $Z \log N + C + 1$. For demonstration purpose, we construct a minimal working example. Let:

$$\vec{y} = ((r, 1, \cdot), (r, 1, \cdot), \dots, (r, 1, \cdot)) \text{ and} \quad (6)$$

$$\vec{y}' = ((r, 1, \cdot), (r, 2, \cdot), \dots, (r, M_L, \cdot)) \quad (7)$$

where r denotes the read operation and \cdot denotes data which is not important for the demonstration. In words, one access sequence consists of M_L accesses to the same element and the second access sequence consists of M_L different accesses to elements $1, 2, \dots, M_L$.

Now, of all the possible sequences $\text{ORAM}(\vec{y})$ can produce, we can see that the sequence $1, 1, \dots, 1$ can be one of them.¹⁸ But, its not hard to see that the same sequence $1, 1, \dots, 1$ can never occur as $\text{ORAM}(\vec{y}')$. The reason for this is simply because we cannot ever map more than M_L elements to the same path (else the Path ORAM invariant is broken i.e. stash overflows) and hence the M_L accesses to the same location cannot all be different elements.

We demonstrate this as an attack on the Path ORAM protocol. Consider a situation where a program is using the Path ORAM protocol to hide its access pattern and we know that the program has the following traits,

$$\text{Access Pattern} = \begin{cases} 1, 1, 1, \dots, 1 & \text{if Secret} = 1 \\ 1, 2, 3, \dots, M & \text{if Secret} = 0 \end{cases}$$

If y is the real access pattern, if ever we see a sequence of M_L or more access made to the same location in $\text{ORAM}(\vec{y})$, we can immediately infer that $\text{Secret} = 1$!¹⁹

6.5 δ bound

Coming back to the Root ORAM protocol, now we can see that the probability of an observed sequence can suddenly jump from 0 to a non-zero value after one data access has been changed. And this is what is captured by the δ in the (ϵ, δ) -differential privacy framework for ORAMs.

Let M_k denote the number $(C + Z(k+1) + 1)$. It is easy to see that there is a sudden jump in the probability from 0 to a non-zero value when the real access is changed at one location when we look at any such sequence. In particular we choose the following two sequences:

$$\begin{aligned} r_1 &= (1, 2, 3, \dots, M_k) \\ r_2 &= (1, 2, 3, \dots, M_k - 1, 1) \end{aligned}$$

If $\Pr[\text{ORAM}(r_i) = o] > 0$ for $i = 1, 2$, then we have already shown the ϵ bound and hence $\delta = 0$. So it remains to find

the smallest probabilities. The same proof goes through for other probability distributions leading to $\epsilon = 2 \log \left(\frac{p_{\max}}{p_{\min}}\right)$.

¹⁸For that matter so can any sequence a, a, \dots, a for any $a \in \{1, 2, 3, \dots, N\}$

¹⁹The reason for this is that there is another constraint in the system which is that no leaf can have more than M_L data blocks mapped to it. This is because each path $P(x)$ has $Z \log N$ buckets and along with the main invariant that each block is stored somewhere along the path from the mapped leaf to the root or in the Stash. We assume that this attack is covered in the failure probability of the ORAM because the probability of this occurring is very very low.

Symbol	Description
p	$1 - 2^{-i}, i = \{1..L\}$
L	From 10 to 21
k	Runs from 1 to L
Z	$Z \in \{2, 3, 4, 5\}$
λ	$\lambda \in \{0.25, 0.5, 0.75, 1, 2, \infty\}$

Table 5: Simulations limits

the maximum δ when one of these terms is 0. WLOG, $\Pr[\text{ORAM}(r_1) = o] = 0$. Hence δ is the maximum value of $\Pr[\text{ORAM}(r_2) = o]$. Now, one simple upper bound on δ can be found by noting the following: Since the probabilities used to compute for each access are at most p_1 (they are either p_1, p_2 or $1/N$ and p_1 is the largest), we can get a quick upper bound on δ as

$$\delta \leq p_1^{M_k} = (1 - p)^{M_k} \quad (8)$$

Where $M_k = (C + Z(k + 1) + 1)$. This completes the δ bound.²⁰ ■

6.6 Bandwidth

Theorem 3. *The bandwidth of the Root ORAM protocol with parameters k, p, Z and λ is $2 \times Z(k + 1) \times (1 + 1/\lambda)$ per real access.*

Proof : The number of blocks in any path of the tree is equal to $Z(k + 1)$ and hence twice the number of blocks are transferred per read and write. Also, they way the parameter λ is set (i.e the way the fake accesses are programmed), we perform on an average λ real accesses per fake access. (the average of a Poisson process with parameter λ is λ). Hence, the bandwidth gets an addition factor of $(1 + 1/\lambda)$ per real access. ■

7. SYSTEMS EVALUATION

In the previous sections, we have established the design space made possible by formalizing DP-ORAM using theoretical analysis. We also saw Root ORAM as a way to achieve some interesting points in the design space. In this section, we analyze the stash size usage, in other words, the outsourcing ratio. We define *outsourced ratio* as the ratio of the total data outsourced to the amount of local storage required.

We resort to simulations to demonstrate the stash size required. We simulate Root ORAM for various values of the parameter to understand the impact of design parameters on the stash size. The parameter choices have been tabulated in Table 5. We begin by giving the details of our implementations.

7.1 Details of the implementation

We implemented the complete functionality of Root ORAM in C++. We plan to make our implementation publicly available as an open source software. We performed all experiments on a 1.4 GHz Intel processor. The Amazon EC2 experiments were performed using a TCP connection for reliable data downloads.

²⁰Given the exponentially decreasing small nature of δ we do not improve on this weak bound.

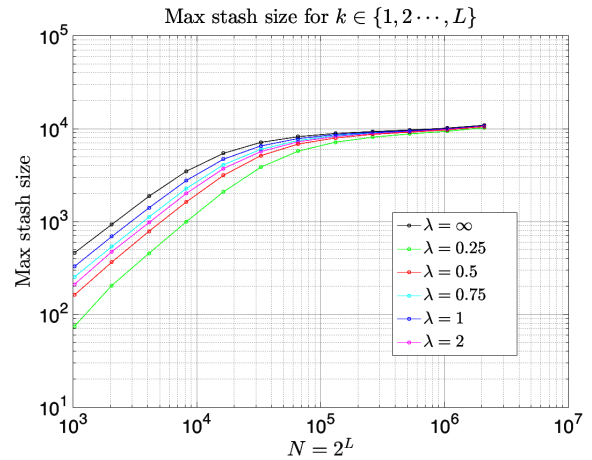


Figure 6: This figure illustrates the maximum stash usage as a function of N . Different lines correspond to different values of λ . To put this in perspective, the maximum stash usage for 10 GB of outsourced data is roughly 40 MB (using a 4 KB block size).

We use random access patterns for the simulations and the maximum stash size is calculated excluding the transient storage for one path. Unlike current work, we independently study the effect of increasing the number of accesses (M) on the max stash size. This is equivalent to giving bounds on the failure probability of the stash. Next we briefly describe the aims of our evaluations before showing its results.

We study the effect of various parameters on the performance. In particular, we study the effect of N on the maximum stash size used (which in other words, captures the failure probability of the ORAM), the dependence of the stash size (outsourcing ratio) on bandwidth and security (ϵ), the growth of the maximum stash size with the number of accesses and finally the latency of Root ORAM protocols for access over remote²¹ Amazon EC2 servers.

7.2 Evaluation results

Max stash usage vs N : In light of the recent paper by Bindschadler *et al.* [2], we base our experimental evaluation by giving due importance to the constants involved. Fig. 6 shows the dependence of the maximum stash used on N , the number of outsourced blocks. Different lines correspond to different values of λ . As can be seen, the effect of λ goes down for relatively large values of N .

To put numbers in perspective, the worst case stash size for standard 4 KB blocks for 10 GB of data outsourced is roughly 40 MB. The growth indicates that this outsourced ratio will be larger for larger values of N .

Outsourcing ratio vs Bandwidth, ϵ : Fig. 7a shows the outsourcing ratio as a function of the bandwidth and security (ϵ) for 4 different values of Z (Z increases from 2 (lowest plane) to 5 (top plane)). When low bandwidth protocols are used (small k), the outsourcing ratio is relatively small. Specifically, in such regimes, for small values of Z , we have an outsourcing ratio of about 10X, whereas for larger Z values, the ratio is almost 1000X. This enables a smartphone client to outsource about 1 TB of data to the untrusted cloud server with less than a 1 GB of local storage

²¹Locations hidden for author anonymity.

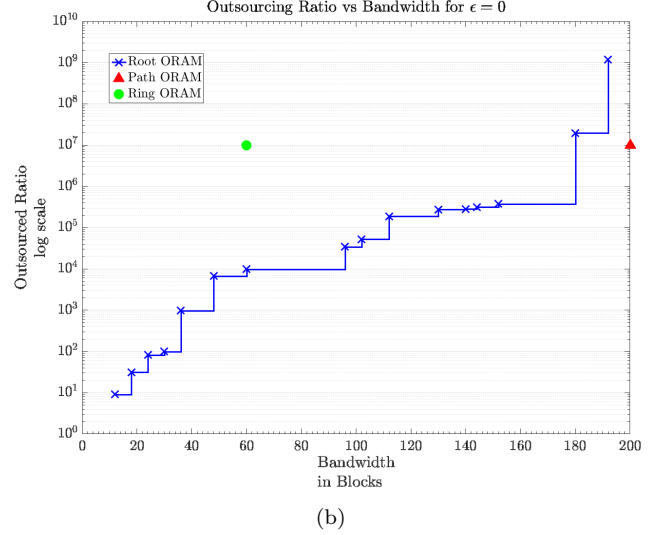
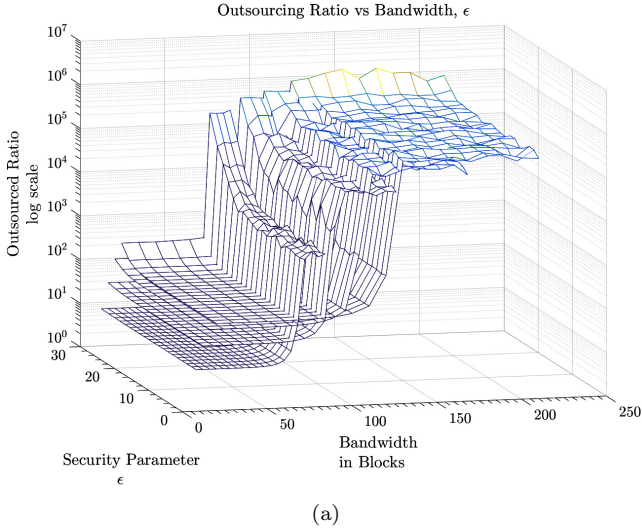


Figure 7: **Fig. 7a** illustrates the outsourcing ratio as a function of k and p . Different surfaces correspond to $Z = 2$ to $Z = 5$ from bottom to top respectively. The y-axis is related to the security parameter p by the equation $p = 1 - 2^{-i}$. **Fig. 7b** shows for a fixed value of ϵ , the trade-off between outsourcing ratio and bandwidth (taking recursion into consideration).

required and extremely low bandwidth requirements. With higher bandwidths, much better outsourcing ratios can be achieved and this can be seen in Fig. 7.

Another interesting feature of Root ORAM is the extremely high outsourcing ratios achieved by models with almost complete tree structures i.e., $k \approx$ from 13 and 20. Though Root ORAM does not have theoretical bounds on the stash usage like other protocols such as Path ORAM or Ring ORAM, it can be seen clearly how these aspects tie together as Path ORAM is a instantiation of Root ORAM in high bandwidth, $\epsilon = 0$ regime, consequently having high outsourcing ratio. Similarly, using appropriate values of p , we can achieve $\epsilon = 0.001, 0.01$ etc. for a given bandwidth with outsourcing ratios as seen in Fig. 7a.

Real-world implementation: Next, we discuss the results of our implementation of Root ORAM over Amazon EC2 servers. Our aim was to compute the latency overhead of a memory accesses as a function of Root ORAM parameters. Fig. 8a depicts latency as a function of the bandwidth for $Z = 5$ while Fig. 8b depicts the latency as a function of the constrained client bandwidth across different values of k . We used the *trickle* application to constrain the bandwidth at client machines to desired values. We notice an order of magnitude difference between latencies at low and high values the bandwidth i.e., low and high values of k .

7.3 Recursive stash reduction

Recursively storing the stash in an ORAM improves the performance exponentially. Suppose that we use Root ORAM with a specific set of parameters k, p, Z and λ and achieves a worst case outsourcing ratio of R . The ϵ, δ values achieved are given by Theorem. 2, bandwidth by Theorem. 3 and the outsourcing ratio by the above simulations.

We can see that using t rounds of recursion, the outsourcing ratio grows exponentially to R^t , whereas the security and the bandwidth increase only linearly viz., $(\epsilon, \delta) \rightarrow (t\epsilon, t\delta)$ and bandwidth $\rightarrow t \times$ bandwidth. This introduces further in-

teresting design points, some of which are shown in Fig. 7b.

7.4 Perfect Security Design Points

Root ORAM offers a number of interesting design points even with perfect security. For $\epsilon = 0$, Root ORAM enables a bandwidth outsourcing ratio trade-off as can be seen in Fig. 7b. For small values of k , Fig. 7a shows that we can design protocols with low bandwidth at the cost of low outsourcing ratio. Similarly, for large values of k , we can achieve extremely high outsourcing ratios at lower bandwidths by using smaller k and utilizing fake accesses. For ex: $k \sim 15, Z = 2, \lambda = 4 \implies 75X$ overhead, a factor of 3X improvement over Path ORAM and comparable to overhead of Ring ORAM. Similarly, Root ORAM can outperform both Path ORAM and Ring ORAM at the cost of a smaller outsourcing ratio as can be seen in Fig. 7b.

7.5 How to choose parameters?

Given the various parameters, we describe how Root ORAM can be enabled for any particular application. As in any other differential privacy application, we set a privacy budget (an upper bound: ϵ_{budget}) for the system. Then we allow ϵ -DP queries till the budget is exhausted i.e., cumulative ϵ of the queries by the composition theorem reaches ϵ_{budget} . Then we can choose two of the following three parameters independently: desired security value ϵ , the available bandwidth and the outsourcing ratio R . The third parameter is determined by the choice of the other two and the optimal choice would be determined by the application requirements.

7.6 Summary

We have shown the practicality of Root ORAM through theoretical analysis, simulations and real-world experiments. Theoretically, we have analyzed the bandwidth and security for different parameter values. Experimentally, we have shown the dependence of the outsourcing ratio and access latency on Root ORAM parameters; demonstrating the fea-

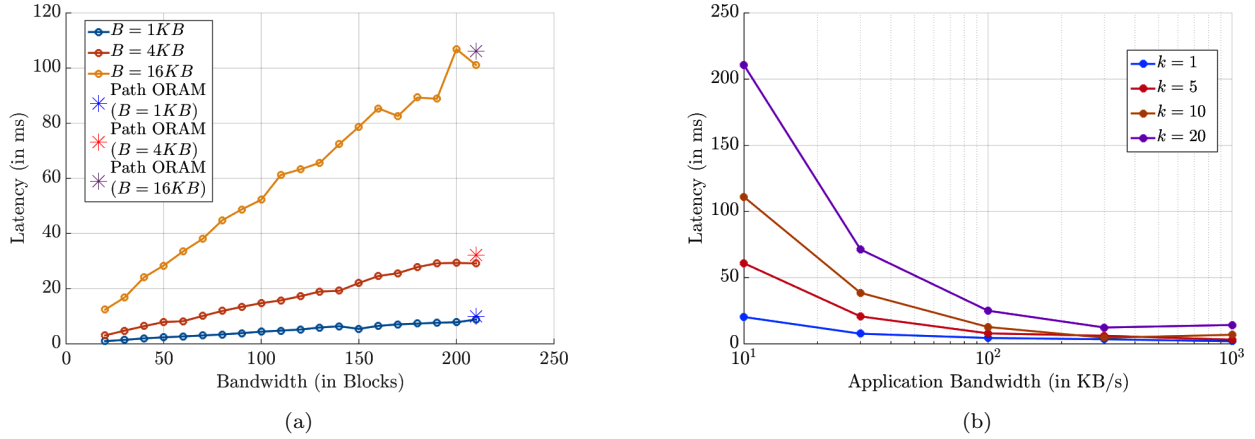


Figure 8: **Real-world implementations over Amazon EC2.** Fig. 8a show the latency as a function the bandwidth for $N = 2^{20}$ and three block sizes viz., 1 KB, 4 KB and 16 KB. Fig. 8b shows the latency as a function of the constrained/limited application bandwidth for 4 KB block sizes. There is a significant difference between the latencies for across k values for constrained bandwidth applications.

sibility of multi-dimensional design space and order of magnitude performance improvement.

8. RELATED WORK

Since the formalization of the concept of an Oblivious RAM, in a seminal paper by Goldreich and Ostrovsky [11], the research community has made substantial progress in making ORAM practical by improving their performance [4, 9, 16, 17, 20–22]. Recent work has also shown the promise of using ORAMs as a critical component in developing protocols for Secure Multi-Party Computation [9].

A recent benchmark for ORAMs has been the Path ORAM protocol [22]. It builds upon previous hierarchical constructions such as [21] and gives theoretical bounds on stash usage. Root ORAM generalizes the construction of Path ORAM to provide a tunable framework offering DP-ORAM guarantees. Root ORAM also reintroduces the eviction scheme of dummy/fake accesses. This was first looked into by Shi *et al.* [19] and formalized in Ren *et al.* [18]. The latter also highlights the potential pitfalls in proposing eviction schemes that are not provably secure while demonstrating the security consequences of one such scheme. Root ORAM avoids that by using dummy accesses indistinguishable from real accesses while also giving rigorous bounds on the security.

Another novel concept that was recently introduced in the ORAM domain was that of the XOR technique to reduce online bandwidth. Online bandwidth, first formalized by Boneh *et al.* in [3] was reduced to $O(1)$ using the XOR technique by Dautrich *et al.* [4]. The XOR technique can be extended to Root ORAM as well, which will further influence the protocol design space.

Two optimizations for [19] were provided by Gentry *et al.* [9]. Concretely, they show the benefits of using a tree structure with multiple leaves instead of 2 as in the case of a binary tree. This idea is in similar spirit as that of Root ORAM though these differ considerably in terms of their working. The higher-degree tree in [9] is a complete higher-degree tree (i.e degree of each node is the same) whereas in the Root ORAM paper, the tree is binary till the last level and only the last level nodes have a higher degree. This

leads to very different dynamics of the two schemes. Similarly, Root ORAM uses fake accesses as its eviction process, different from [9].

ORAM has been implemented and shown to be feasible at a chip level in prototypes such as the Ascend architecture [8] and the Phantom architecture [14]. But unlike the case of chip-level implementations where trusted local cache is expensive, most other applications have more client space. In fact, in today’s settings, it is feasible to have client storage of the order of 1 GB for outsourced data of about 1 TB [2].

In short, Root ORAM is the only protocol with tunable security-bandwidth-outsourcing ratio construction. Similarly, none of the previous works deal with statistical privacy in the ORAM context, hence the formalization of differentially private ORAMs is an important contribution of this work.

9. LIMITATIONS AND FUTURE WORK

To enable the design of practical ORAM schemes for applications with stringent bandwidth constrain, it is desirable to have statistical privacy and Root ORAM demonstrates the first step in this direction by introducing a tunable framework that provides rigorous differential privacy guarantees. This opens up a number of research ideas which remain unexplored in the current work.

First, we would like to explore the integration of techniques that leverage server-side computation in the Root ORAM architecture, such as the XOR technique [3,4]. Such an approach can trade-off bandwidth at the cost of server-side computation and can further influence the design space of differentially private ORAMs. Second, we would like to explore the effect of varying the ratio of number of blocks outsourced to the size of the server-side storage. Gentry *et al.* [9] have explored similar techniques in the case of Path ORAM, and it would be interesting to combine these techniques with Root ORAM.

Our experimental results have demonstrated the required stash size for various parameters of Root ORAM. However, we acknowledge that Root ORAM lacks rigorous theoretical guarantees on stash usage. In future work, it would be interesting to rigorously bound the stash usage of Root ORAM.

10. CONCLUSIONS

To summarize, we present Root ORAM, a tunable family of ORAM protocols which provide a multi-dimensional trade-off between bandwidth (performance), security and local storage requirements. We introduce and formalize the notion of a differentially private ORAM, which to our knowledge is the first of its kind.

We evaluate the protocol using theoretical analysis, simulations, and real world implementation on Amazon EC2. We theoretically prove that Root ORAM provides the rigorous privacy guarantees of differential privacy. We experimentally demonstrate that the stash size used by Root ORAM is bounded for realistic values of parameters. Overall, Root ORAM can serve as an enabler for real-world deployment of oblivious RAM by providing novel design points that provide an order of magnitude performance improvement over current state-of-the-art.

11. REFERENCES

- [1] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alexander Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 294–304. ACM, 1993.
- [2] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: The gap, the fallacy, and the new way forward. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, pages 837–849, New York, NY, USA, 2015. ACM.
- [3] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote oblivious storage: Making oblivious RAM practical. 2011.
- [4] Jonathan Dautrich, Emil Stefanov, and Elaine Shi. Burst ORAM: Minimizing ORAM response times for bursty access patterns. In *USENIX Security*, 2012.
- [5] Jonathan L Dautrich Jr and Chinya V Ravishankar. Compromising privacy in precise query protocols. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 155–166. ACM, 2013.
- [6] Cynthia Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.
- [7] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503. Springer, 2006.
- [8] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pages 3–8. ACM, 2012.
- [9] Craig Gentry, Kenny A Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation. In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2013.
- [10] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC '87*, pages 182–194, New York, NY, USA, 1987. ACM.
- [11] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.
- [12] MS Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proc. NDSS*, volume 14, 2014.
- [13] Yingbin Liang, H Vincent Poor, et al. Information theoretic security. *Foundations and Trends in Communications and Information Theory*, 5(4–5):355–580, 2009.
- [14] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiawicz, and Dawn Song. Phantom: Practical oblivious computation in a secure processor. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 311–324. ACM, 2013.
- [15] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.
- [16] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association.
- [17] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Ring ORAM: Closing the gap between small and large client storage oblivious RAM. Technical report, Cryptology ePrint Archive, Report 2014/997, 2014. <http://eprint.iacr.org>.
- [18] Ling Ren, Xiangyao Yu, Christopher W Fletcher, Marten Van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious RAM in secure processors. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 571–582. ACM, 2013.
- [19] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $o((\log n)^3)$ worst-case cost. In *Advances in Cryptology-ASIACRYPT 2011*, pages 197–214. Springer, 2011.
- [20] Emil Stefanov and Elaine Shi. Oblivstore: High performance oblivious cloud storage. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 253–267. IEEE, 2013.
- [21] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652*, 2011.
- [22] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.